

This is available online in html, pdf and postscript, at
<http://www.cs.bham.ac.uk/research/projects/cogaff/00-02.html#71>
(A slightly shorter version appeared in *Computing in Schools*, 1981.)

EXPERIENCING COMPUTATION

A Tribute to Max Clowes

by

Aaron Sloman¹

(Reader in Philosophy and Artificial Intelligence)

Cognitive Studies Programme²

School of Social Sciences,

The University of Sussex

1 Introduction

Max Clowes died of a heart attack on Tuesday 28th April 1981. He was one of the best known British researchers in Artificial Intelligence, having done pioneering work on the interpretation of pictures by computers. His most approachable publication is cited below. He was an inspiring teacher and colleague, and will be remembered for many years by all who worked with him. He helped to found AISB, the British society for the study of Artificial Intelligence and the Simulation of Behaviour, now expanded to a European society. This tribute is concerned mainly with his contribution to education.

He was one of the founder members of the Cognitive Studies Programme begun in 1974 at the University of Sussex, a novel attempt to bring together a variety of approaches to the study of Mind, namely Psychology, Linguistics, Philosophy and Artificial Intelligence. During the last few years his interests centred mainly on the process of teaching computing to absolute beginners, including those without a mathematical or scientific background. He was one of the main architects of the

¹Now at the University of Birmingham <http://www.cs.bham.ac.uk/~axs/>. I discovered an electronic version of this article among some old files recently and thought it would be useful to make it available online, at least for people who remember Max, and perhaps others. I have added a few footnotes, putting the text in the context of subsequent developments.

²The Cognitive Studies Programme, was founded by Max Clowes and others at the University of Sussex as a cross-disciplinary undergraduate programme within the School of Social Studies. Later, a few years after he died it became a separate School of Cognitive Sciences. Within a few years it had enlarged further and become what is now known as COGS, The School of Cognitive and Computing Sciences, with an international reputation. But for the early inspiration influence of Max Clowes it would never have existed. My own role in its development was a direct consequence of the influence Max had in changing my research direction, from about 1969

Sussex University POP11 teaching system (along with Steve Hardy and myself), which has gradually evolved since 1975. In this brief tribute, I shall sketch some main features of the system, and hint at the unique flavour contributed by Max.

POP11 embodies a philosophy of computer education which is relatively unusual. It includes a language, a program-development environment, and a collection of teaching materials including help facilities, much on-line documentation, and a large collection of exercises and mini-projects. Unfortunately, it is at present available only on a PDP11 computer running the Unix operating system, though a version now being written in C should be available for use on a VAX by the end of this year.³

When we started planning the system, in 1974, we were much influenced by the writings of John Holt (see references at end), the work on LOGO at MIT by Seymour Papert and colleagues, and at Edinburgh University by Sylvia Weir, Tim O'Shea, and Jim Howe.

These influenced our conviction that learners of all ages should be treated not like pigeons being trained by a schedule of punishment and reward, but like creative scientists driven by deep curiosity and using very powerful cognitive resources. This entailed that learners should not be forced down predetermined channels, but rather provided with a rich and highly structured environment, with plenty of opportunities to choose their own goals, assess their achievements, and learn how to do better next time through analysis of failures.

Although these needs can to some extent be met by many older learning environments (e.g. meccano sets, learning a musical instrument, projects), the computer seemed to be potentially far more powerful, on account of its speed, flexibility, reactivity and ability to model mental processes. Instead of making toy cranes or toy aeroplanes, or dolls, students could make toy minds.

Unlike educational philosophies which stress 'free expression', this approach stresses disciplined, goal oriented, technically sophisticated, activities with high standards of rigour: programs will not work if they are badly designed. Yet the computer allows free expression to the extent that students can choose their own goals, and their own solutions to the problems, and the computer will patiently, more patiently than any teacher, pay detailed attention to what the student does, and comment accordingly, by producing error messages, or running the program and producing whatever output is required. Of course, error messages need to be

³Note added 2001: In fact the version in C proved to be a temporary bridge towards a Pop11 in Pop11 compiler subsequently developed by John Gibson, which became the core of the multi-language Poplog system, which was later sold world wide. It is now available free of charge with full system sources at <http://www.cs.bham.ac.uk/research/poplog/freepoplog.html> where much of the still useful AI teaching material has its roots in his approach to teach. E.g. <http://www.cs.bham.ac.uk/research/poplog/teach/river>

far more helpful than in most programming environments, and the system should make it easy for the student to make changes, to explore ‘where the program has got to’, and to try out modifications and extensions without a very lengthy and tedious edit compile and run cycle.

These ideas are embodied in a course, *Computers and Thought*, offered as an unassessed optional first year course for students majoring in Humanities and Social Science subjects. By making the computer do some of the things people can do, like play games, make plans, analyse sentences, interpret pictures, the students learn to think in a new way about their own mental processes. Max put this by saying that he aimed to get students to ‘experience computation’ and thereby to ‘experience themselves as computation’.⁴ In other words, our answer to the student’s question ‘Does that mean I’m a computer?’ is ‘Yes’. Of course people are far more intricate, varied, flexible, and powerful than any man-made computer. Yet no other currently available framework of concepts is powerful enough to enable us to understand memory, perception, learning, creativity and emotions.

2 Choosing a language

Part of the LOGO philosophy was that beginners playing with computers needed a very powerful language, making it easy to perform interesting tasks quickly. (See Papert). Interesting tasks include building ‘toy minds’. Thus we needed a language which is interactive and provides procedures with recursion and local variables, and facilities for non-numerical problem solving, such as list manipulation. This rules out BASIC. LOGO is much more powerful, but, we felt, did not go far enough: after all, it was designed for children, so it could not be powerful enough for children! PASCAL was ruled out as too unfriendly and even less powerful than LOGO. (E.g. the type structure makes it impossible to program a general-purpose list-processing package: the package has to be re-implemented for numbers, words, lists etc.) We did not know about APL. It might have been a candidate, though its excessively compressed syntax which delights mathematicians is hardly conducive to easy learning by the mathematically immature. Moreover it appears to have been designed primarily for mathematical applications, and does not seem to have suitable constructs and facilities for our purposes. PROLOG would have been considered had an implementation been available, though it too is geared too much towards a particular class of problems, and is hard to use for

⁴Some years later, ideas that had developed out of the course appeared in a book: Sharples, M. Hogg, D. Hutchison, C. Torrance, S. Young, D. *Computers and Thought: a practical introduction to Artificial Intelligence* MIT Press 1989. Many students still find this book an excellent route into AI and computational cognitive science.

others.

We therefore reduced the choice to LISP and POP2, and settled for the latter because of its cleaner, more general semantics (e.g. functions are ordinary values of variables), more natural syntax, and convenient higher-level facilities such as partial-application and powerful list-constructors (Burstall et. al). A subset of POP2 was implemented on a PDP11/40 by Steve Hardy, and then extended to provide a pattern-matcher, database, and other useful features. We now feel that the power of PROLOG (see Kowalski 1979) should be available as a subsystem within POP, and are planning extensions for the VAX version.⁵

3 More than just a language

From the start we intended to provide a wide range of facilities in the library, so that students could easily write programs to do things which interested them: draw pictures, analyse pictures, play games, have conversations relating to a database of knowledge, etc. We soon also found the need for help-facilities, on-line documentation of many kinds, and a simple, non authoritarian teaching program (written in POP11, and calling the compiler as a subroutine to execute the student's instructions), which could, for some learners, feel less daunting than a twenty page printed handout.

One of the ideas that played an important role in our teaching strategy was an analogy between learning a programming language, and learning a natural language, like English. The latter does not require formal instruction in the syntax and semantics of the language. The human mind seems to possess very powerful capacities for absorbing even a very complex formalism through frequent and fruitful *use*. So, instead of starting with lectures on the language, we decided to give the students experience of using the language to get the computer to do things we hoped would make sense to them. So they were encouraged to spend a lot of time at the terminal, trying out commands to draw pictures, generate sentences, create and manipulate lists, etc, and as they developed confidence, to start working towards mini-projects. Extending or modifying inadequate programs produced by the teacher (or other students) provides a means of gaining fluency without having to build up everything from the most primitive level. Naturally, this did not work for everyone. Some preferred to switch at an early stage to learning from more formal documentation. Some found the pain of even minor errors and failures too traumatic and needed almost to be dragged back to try again - often with

⁵Chris Mellish implemented a Prolog in Pop-11 in 1981, and after that the system combining both languages came to be known as Poplog. Later, other languages were added.

some eventual success. Some, apparently, had insuperable intellectual limitations, at least within the time-scales available for them to try learning to program. But many students found it a very valuable mind-stretching experience.

One of the ways in which Max contributed to this was his insistence that we try to select approaches and tasks which were going to be more than just a trivial game for the student. He was able to devise programming exercises which could be presented as powerful metaphors for important human mental processes - such as the pursuit of goals, the construction of plans, the perception and recognition of objects around us and the interpretation of language. He would start the 'Computers and Thought' course by introducing students to a simple puzzle-solving program and erect thereon a highly motivating interpretation: treating it as a microcosm of real life, including the student's own goal-directed activities in trying to create a working program. (Perhaps this is not unconnected with a slogan he defended during his earlier work on human and machine vision: "Perception is controlled hallucination" - hallucinating complex interpretations onto relatively simple programs helps to motivate the students and give them a feel for the long term potential of computing).

Moreover, he always treated teaching and learning as more than just an intellectual process: deep emotions are involved, and need to be acknowledged. So he tried to help students confront their emotions of anxiety, shame, feeling inadequate, etc., and devised ways of presenting material, and running seminars, which were intended to help the students build up *confidence* as well as understanding and skills. There is no doubt that for many students the result was an unforgettable learning experience. Whether they became expert programmers or not, they were changed persons, with a new view of themselves, and of computing. Some of this was due to his inimitable personality. In addition he advocated strategies not used by many university teachers at any rate: such as helping all the students in a class to get to know one another, prefacing criticisms with very encouraging comments, and helping students cope with their own feelings of inadequacy by seeing that others had similar inadequacies and similar feelings about them, whilst accepting that such 'bugs' were not necessarily any more permanent than bugs in a computer program.

As a teacher I found myself nervously treading several footsteps behind him - too literal-minded to be willing to offer students his metaphors without qualification, yet benefitting in many ways from his suggestions and teaching practice.

Just before he left Sussex we had a farewell party, at which he expressed the hope that we would never turn our courses into mere computer science. There is little risk of that, for we have learnt from him how a different approach can inspire

and motivate students. The computer science can come at a later stage - for those who need it. For many, who may be teachers, managers, administrators, etc. rather than programmers or systems analysts, the formalities of computer science are not necessary. What is necessary is a good qualitative understanding of the range of types of things that can be done on computers, and sufficient confidence to face a future in which computation in many forms will play an increasingly important role.

None of this should be taken as a claim that the teaching system based on POP11, used by Max and the rest of us, is anywhere near perfect. We are aware of many flaws, some of which we feel we can remedy. But there is still a great deal of exploring to be done, in the search for a good learning environment, and good learning experiences. Moreover, we don't know how far what is good for novice Arts university students would also be good for school children, though several have played with our system and enjoyed it. We are still in the process of improving the POP virtual machine to make it a more natural tool for thinking about processes. This is a never-ending task. Probably a language is needed which can be 'disguised' to present a simpler interface for the youngest learners, without sacrificing the power to do interesting things very quickly. To some extent the 'macro' facility in POP (see Burstall et. al.) makes this possible.

4 Computing in Schools?

In December 1980 Max left Sussex, to join a project on computing in schools. Although I don't know exactly what his plans were, I feel that he would probably have fed many important new ideas into the educational system. New ideas are surely needed, for teaching children to program in BASIC is like teaching them to climb mountains with their feet tied together: the permitted steps are so very small. Moving towards COMAL will merely loosen the ropes a little. Teaching them PASCAL will loosen the ropes a lot more, but add heavy weights to their feet: taking some big steps is possible in PASCAL, but unnecessarily difficult. And some things are not possible, as mentioned previously.

The problems of providing better teaching languages and teaching environments are enormous, since available computers are much too small, and there is too little expertise on tap in schools. I think Max might have tried to press for funds to be diverted from stand-alone microcomputers to larger, shared, machines, or networks, making available better languages, shared libraries, larger address spaces and increased opportunities for learners to help one another, including teachers. This may be more expensive: but what could be a more important use of

computers?

Such a system, based on a DEC 11/23, running the UNIX operating system, has just been introduced at Marlborough College, partly as a result of Max's inspiration. It will be interesting to watch what happens there. Maybe we'll learn from that how to use the newer, cheaper, bigger, faster systems that will shortly be on the market. Let us hope the existing puny machines with their inadequate languages will not have turned too many people off computing for life.

Acknowledgements

I am grateful to Reg Baukham and Masoud Yazdani, who reminded me of some features of Max's teaching omitted from a first draft, and who pointed me at things he had written in CSGAS, an informal magazine edited by Masoud Yazdani.

References

R M Burstall, et. al. *Programming in POP2*, Edinburgh University Press, 1972

Max Clowes, 'Man the creative machine: A perspective from Artificial Intelligence research', in J. Benthall (ed) *The Limits of Human Nature*, Allen Lane, 1972.

John Holt, *How Children Learn*, Penguin Books

John Holt, *How Children Fail*, Penguin Books.

Robert Kowalski *Logic for Problem Solving*, North Holland, 1979.

Seymour Papert, *Mindstorms*, Harvester Press, 1981.

Entry for Max Clowes at HOPL (History of Programming Languages) web site:

<http://hop1.murdoch.edu.au/showlanguage2.prx?exp=7352>